Relatório de Análise – Simulador de Escalonador de Processos

1. Justificativa de Design

A estrutura de dados utilizada é uma **Fila**, implementada com uma **Lista Ligada** que mantém ponteiros para o início (`cabeca`) e o fim (`cauda`). Esta escolha é altamente eficiente para o comportamento de um escalonador porque as operações primárias — adicionar um processo ao final da fila de prontos (`enqueue`) e remover o próximo processo a ser executado do início (`dequeue`) — são realizadas em tempo constante, **O(1)**. Não há necessidade de percorrer a lista, garantindo máxima performance para as operações mais frequentes.

2. Análise de Complexidade (Big-O)

A eficiência das operações fundamentais se reflete na complexidade geral:

- **`addFinal(processo)` (Adicionar na Fila):** `O(1)` - A inserção ocorre diretamente no final usando o ponteiro `cauda`.
- **`removerProcesso()` (Remover da Fila):** `O(1)` - A remoção ocorre diretamente do início usando o ponteiro `cabeca`.
- **`executarCicloDeCPU()` (Ciclo do Escalonador):** `O(1)` - O ciclo consiste em um número fixo de verificações e operações de fila (`add` e `remove`), que são todas `O(1)`. A complexidade não aumenta com o número de processos nas filas.

3. Análise da Anti-Inanição

A lógica garante justiça ao evitar a **inanição (starvation)**. Um contador monitora as execuções seguidas de processos de alta prioridade. Ao atingir um limite (5), o escalonador força a execução de um processo de prioridade menor, "quebrando" o monopólio da fila de alta prioridade e dando chance aos demais.

Risco sem essa regra: Se houvesse um fluxo contínuo de processos de alta prioridade, os processos de média e baixa prioridade jamais seriam executados. O sistema se tornaria injusto e não responsivo para tarefas menos prioritárias.

4. Análise do Bloqueio (Ciclo de Vida do Processo "DISCO")

1. **Entrada:** O processo é adicionado à sua fila de prontos (ex: `listaAltaPrioridade`).
2. **Execução e Bloqueio:** Ao ser escolhido para executar, o scheduler detecta a necessidade do recurso "DISCO". O processo é imediatamente movido para a `listaBloqueados`.
3. **Espera:** Ele permanece na `listaBloqueados` por exatamente um ciclo de CPU.
4. **Desbloqueio:** No início do ciclo seguinte, ele é removido da `listaBloqueados` e reinserido no final de sua fila de prontos original.

5. **Retorno:** O processo volta a competir pela CPU. Na sua próxima execução, o recurso não será mais solicitado, e ele executará seu ciclo de trabalho normalmente.

5. Ponto Fraco e Melhoria Teórica

- **Ponto Fraco:** O principal gargalo é o **modelo de bloqueio simplista**. Assumir que toda operação de I/O dura exatamente **1 ciclo de CPU** é irrealista. Operações de disco podem variar drasticamente em tempo. O sistema não simula uma espera real, apenas impõe uma penalidade fixa de um ciclo.
- **Melhoria Teórica:** Implementar um **desbloqueio baseado em eventos ou timers**. Ao bloquear, um processo receberia um "tempo de bloqueio" (ex: 5 ciclos). O scheduler, a cada ciclo, decrementaria esse timer para todos os processos bloqueados. Um processo só voltaria à fila de prontos quando seu timer chegasse a zero. Isso simularia durações de I/O variáveis e criaria um comportamento de bloqueio muito mais realista e complexo.